⚛️ React Hook: `useEffect` Notes

🔷 Basic Syntax

```
useEffect(() ⇒ {
  // side-effect code here

  return () ⇒ {
    // cleanup code here (optional)
  };
}, [dependencies]);
```

💡 What is `useEffect`?

- `useEffect` lets you perform **side effects** in function components (like API calls, event listeners, timers, etc.).
- It runs **after the component renders**.
- You can also do **cleanup** before the component unmounts or before the effect runs again.

📘 Variations of `useEffect`

✅ Variation 1: Runs on every render

```
useEffect(() ⇒ {
  alert('I will run on each render');
});
```

- No dependency array → effect runs **after every render**.
- 🔁 Triggered even if **nothing changed**.

✅ Variation 2: Runs when a specific state changes

```
useEffect(() ⇒ {
  alert('Count changed');
}, [count]);
```

- Only runs when `count` changes.
- 📌 `count` is a **dependency**.

✅ Variation 3: Runs only on first render (Mount)

```
useEffect(() ⇒ {
  alert('This runs only on first render');
}, []);
```

- Empty dependency array `[]` → runs **only once**.
- Useful for initializing data (e.g., API calls).

✅ Variation 4: Runs when multiple states change

```
useEffect(() ⇒ {
  alert('Count or Total changed');
}, [count, total]);
```

- Runs when **either** `count` or `total` is updated.
- Dependency array can have **multiple values**.

♻️ Cleanup Function (Unmount / Re-run)

```
useEffect(() ⇒ {
  alert('Count is updated');

  return () ⇒ {
    alert('Count is unmounted from UI');
  };
}, [count]);
```

- Cleanup function is used to **remove side effects**.
- Runs:
  - Before the next effect re-runs (if `count` changes again).
  - When the component is **unmounted**.

🧠 Notes & Best Practices

- Always declare **only those dependencies** in the array which are used inside the effect.
- Missing dependencies can lead to **unexpected bugs**.
- If cleanup is skipped for things like timers or subscriptions, it can lead to **memory leaks**.

🧪 Example States Used

```
const [count, setCount] = useState(0);
const [total, setTotal] = useState(0);
```

- `countHandle` → Increments count.
- `totalHandle` → Increments total.
- You can use useEffect to monitor both independently or together.

📦 Custom Component Use Case

```
<Windowwidth />
```

- A good example of **component-based logic** separation.

- You can also use `useEffect` inside custom components like `Windowwidth` to track window resizing, etc.
